# JavaScript - Uma linguagem de script dinâmica para aplicações cliente

# 1. Primeiros Passos Com JavaScript

## 1.1. O que é JavaScript

As APIs proveem muitas funcionalidades ao JS.

Geralmente são divididas em duas categorias:

```
*APIs de navegadores*: já vem implementadas no navegador, e são capazes de expor dados do ambiente do computador, ou fazer coisas complexas e úteis.
```

```
*APIs de terceiros*: não estão implementados no navegador automaticamente.
```

## 1.2. O que JavaScript está fazendo na sua página web?

### 1.2.1. Ordem de Execução do JavaScript

Normalmente é executado de cima para baixo.

### 1.2.2. Código Interpretado x compilado

O JS é uma linguagem interpretada, a compilação é manipulada em tempo de execução, e não antes.

### 1.2.3. Lado do servidor x Lado do Cliente

### 1.2.4. Código dinâmico x estático

**Código dinâmico** atualiza a exibição de uma página web/app para mostrar coisas diferentes em circunstâncias diferentes, gerando novo conteúdo como solicitado.

**Código estático** só mostra o mesmo conteúdo o tempo todo.

## 1.3. Como adicionar JS em uma página

Utiliza-se o elemento <script>.

### 1.3.1. Javascript interno

Exemplo em: PrimeiroPassosComJavaScript/applyjs.html

### 1.3.2. JS externo

É utilizado um arquivo .js com os algoritmos, e não colocado diretamente na página html.

Exemplo em: PrimeiroPassosComJavaScript/applyjs.html

### 1.3.3. Manipuladores de JS inline

incluir funções, métodos, algoritmos diretamente nos elementos html.

**É uma má prática de desenvolvimento!**

### 1.3.4. Estratégias para o carregamento de scripts

Um problema comum é que todo o HTML de uma página é carregado na ordem em que ele aparece. Se estiver usando JS para manipular alguns elementos da página (sendo mais preciso, manipular o DOM), o código não irpa funcionar caso o JS for carregado e executado antes mesmo dos elementos HTML estarem disponíveis.

Algumas soluções para isso são exemplificadas abaixo.

#### Ouvidor de eventos

No exemplo do código JS colocado de maneira interna no html, pode-se colocar o algoritmo em volta deste código:

```
document.addEventListener("DOMContentLoaded", function () {
    ...
});
```

Este código é um *event listener* (ouvidor de eventos), que ouve e aguarda o disparo do evento "DOMContentLoaded" vindo do browser, evento que significa que o corpo do HTML está completamente carregado e pronto. O código JS que estiver dentro desse bloco não será executado até que o evento seja disparado, portanto, o erro será evitado.

No **exemplo de código JS externo**, para resolver o problema advindo do não carregamento da página html, utiliza-se o atributo `defer`, que informa ao browser para continuar renderizando o conteúdo HTML uma vez que a tag `<script>` foi atingida.

```
<script src="script.js" defer></script>
```

Neste caso, o script e o HTML irão carregar de forma simultânea e o código irá funcionar.

**async e defer**

Esses dois atributos podem ser usados para evitar o problema com o bloqueio de scripts.

**async** = os scripts que são carregados usando este atributo irão baixar o script sem bloquear a renderização da página e irão executar imediatamente após o script terminar de ser disponibilizado. Não garante que os scripts carregados rodem em uma ordem específica, mas garante que eles impeçam o carregamento do restante da página. Seu melhor uso é quando os scripts de uma página rodam de forma independente entre si e também não dependem de nenhum outro script.

```
<script async src="js/vendor/jquery.js"></script>
<script async src="js/script2.js"></script>
<script async src="js/script3.js"></script>
```

No exemplo acima, não dá para garantir que o script. jquery.js carregará antes ou depois do script2.js e script3.js . Nesse caso, se alguma função desses scripts dependerem de algo vindo do jquery, ela produzirá um erro pois o jquery ainda não foi definido/carregado quando os scripts executaram essa função.

async deve ser usado quando houver muitos scripts rodando em background, e você precisa que estejam disponíiveis o mais rápido possível.

**defer**: os scripts com o atributo defer irão carregar na ordem que aparecem na página.

Resumindo:

- async e defer istruem o browser a baixar os scripts numa thread (processo) à parte, enquanto o resto da página (o DOM, etc.) está sendo baixado e disponibilizado de forma não bloqueante.
- Se os seus scripts precisam rodar imediatamente, sem que dependam de outros para serem executados, use async.
- Se seus scripts dependem de outros scripts ou do DOM completamente disponível em tela, carregue-os usando defer e coloque os elementos <script> na ordem exata que deseja que sejam carregados.

### 1.3.5. Comentários

De uma linha //

Multiplas linhas

/* Seu texto aqui */

# 2. Um Mergulho no JavaScript

## 2.1. Pensando como um Programador

## 2.2. Exemplo - Jogo adivinhe um número

Vamos imaginar que o seu chefe te deu as seguintes diretrizes para criar este jogo:

```
Quero que você crie um jogo simples do tipo adivinhe um número. Ele deve gerar um
número aleatório de 1 a 100, depois desafiar o jogador a adivinhar o número em 10
rodadas. A cada rodada deve ser dito ao jogador se ele está certo ou errado, se
estiver errado, deve ser dito se o palpite é muito baixo ou muito alto. Também deve
ser mostrado ao jogador os números que ele tentou adivinhar anteriormente. O jogo
termina se o jogador acertar o número ou acabarem o número de tentativas. Quando o
jogo acabar, deve ser dado ao jogador a opção de jogar novamente.
```

Olhando para o enunciado, a primeira coisa que devemos fazer é quebrá-lo em pequenas tarefas, da forma mais parecida com o pensamento de um programador quanto possível:

1. Gerar um número aleatório entre 1 e 100.
2. Gravar o número do turno que o jogador está. Iniciar em 1.
3. Dar ao jogador uma forma de adivinhar o número.
4. Após a tentativa ter sido submetida, primeiro gravar em algum lugar para que o usuário possa ver as tentativas anteriores.
5. Depois, verificar se o palpite está correto.
6. Se estiver correto: Escrever mensagem de parabéns. Impedir que o jogador insira mais respostas (isso pode bugar o jogo). Mostrar controle que permita ao jogador reiniciar o jogo.
7. Se o palpite estiver errado e o jogador ainda tem turnos sobrando: Dizer ao jogador que ele está errado. Permitir que ele insira outra resposta. Incrementar o número do turno em 1.
8. Se o jogador está errado mas não tem turnos sobrando: Dizer ao jogador que o jogo acabou. Impedir que o jogador insira mais respostas (isso pode bugar o jogo). Mostrar controle que permita ao jogador reiniciar o jogo.
9. Quando reiniciar, tenha certeza de resetar todas as variáveis e a interface do jogo, então volte para o passo 1.

### 2.2.1. Configuração inicial

Baixar html de exemplo em https://github.com/mdn/learning-area/blob/main/javascript/introduction-to-js-1/first-splash/number-guessing-game-start.html

### 2.2.2. Adicionando variáveis para armazenas nossos dados

```
var numeroAleatorio = Math.floor(Math.random() * 100) + 1;

var palpites = document.querySelector(".palpites");
var ultimoResultado = document.querySelector(".ultimoResultado");
var baixoOuAlto = document.querySelector(".baixoOuAlto");
```

```
var envioPalpite = document.querySelector(".envioPalpite");
var campoPalpite = document.querySelector(".campoPalpite");

var contagemPalpites = 1;
var botaoReinicio;
```